

Java: Basilari di Gerarchie di Classi

Sommario: 2 Maggio, 2019

- Classi e Sottoclassi:
Oggetti, Costruttori e il costrutto **new**
- Sottoclassi:
Interfacce, Ereditarietà e Shadowing
- Overriding di metodi
- Binding dinamico dei metodi:
Late Binding
- Overloading e Overriding:
Rischi e Cautele.
- Esercizi

Classi e Sottoclassi: Oggetti e Costruttori

- **Sottoclasse** Estende la gerarchia delle classi del programma:
 - Estende campi e metodi della (super)classe
 - Eredita campi e metodi della superclasse

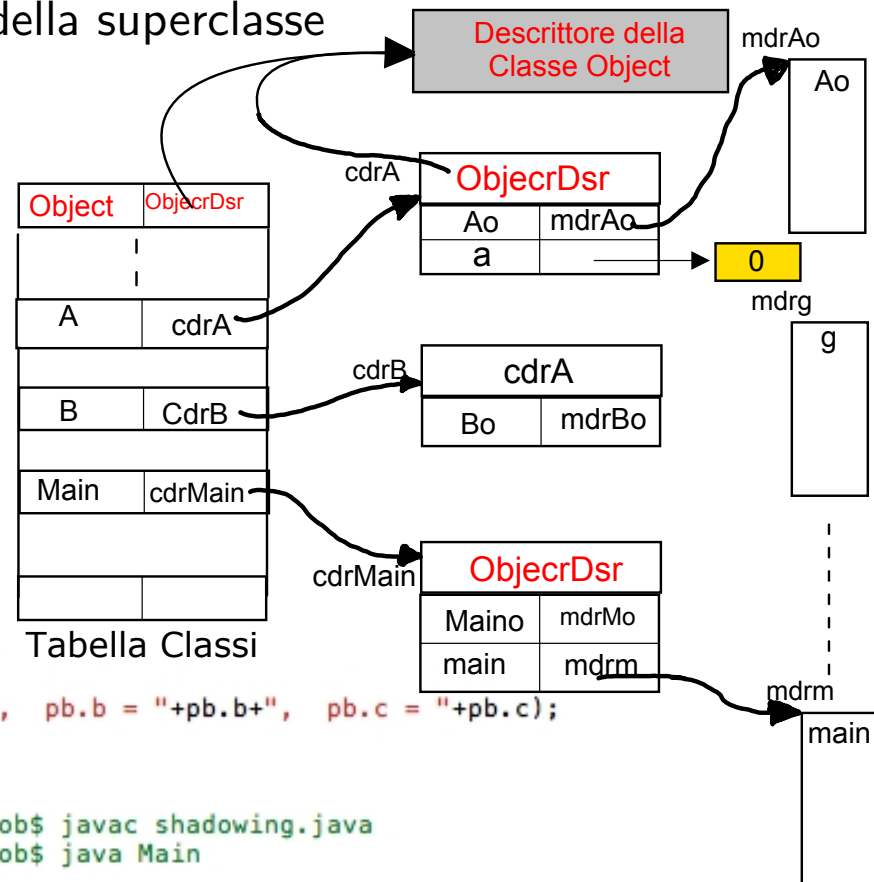
```
class A{
    static int a;
    int b;
    void g(){};
}
class B extends A{
    int c;
    void h(){b=(a=5)+(c=2);}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        pb.h();
        System.out.println("pb.a = "+pb.a+", pb.b = "+pb.b+", pb.c = "+pb.c);
    }
}
/*
Marco-Bellias-MacBook-Pro:sottoclasse marcob$ javac shadowing.java
Marco-Bellias-MacBook-Pro:sottoclasse marcob$ java Main
pb.a = 5, pb.b = 7, pb.c = 2
Marco-Bellias-MacBook-Pro:sottoclasse marcob$
*/
```

Classi e Sottoclassi: Oggetti e Costruttori

- **Sottoclasse** Estende la gerarchia delle classi del programma:
 - Estende campi e metodi della (super)classe
 - Eredita campi e metodi della superclasse

(a destra: vista delle classi)

```
class A{
    static int a;
    int b;
    void g(){};
}
class B extends A{
    int c;
    void h(){b=(a=5)+(c=2);}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        pb.h();
        System.out.println("pb.a = "+pb.a+", pb.b = "+pb.b+", pb.c = "+pb.c);
    }
}
/*
Marco-Bellias-MacBook-Pro:sottoclasse marcob$ javac shadowing.java
Marco-Bellias-MacBook-Pro:sottoclasse marcob$ java Main
pb.a = 5, pb.b = 7, pb.c = 2
Marco-Bellias-MacBook-Pro:sottoclasse marcob$
*/
```

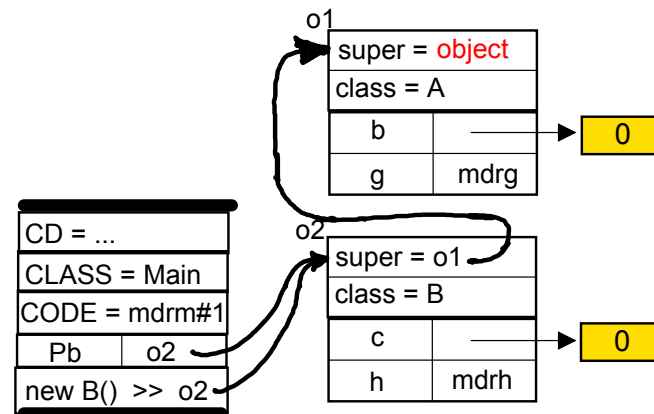


Classi e Sottoclassi: Oggetti, Costruttori e il costrutto new

- **Sottoclasse** Estende la gerarchia delle classi del programma:
 - Gli oggetti estendono campi e metodi degli oggetti della super
 - Gli oggetti ereditano campi e metodi degli oggetti della super

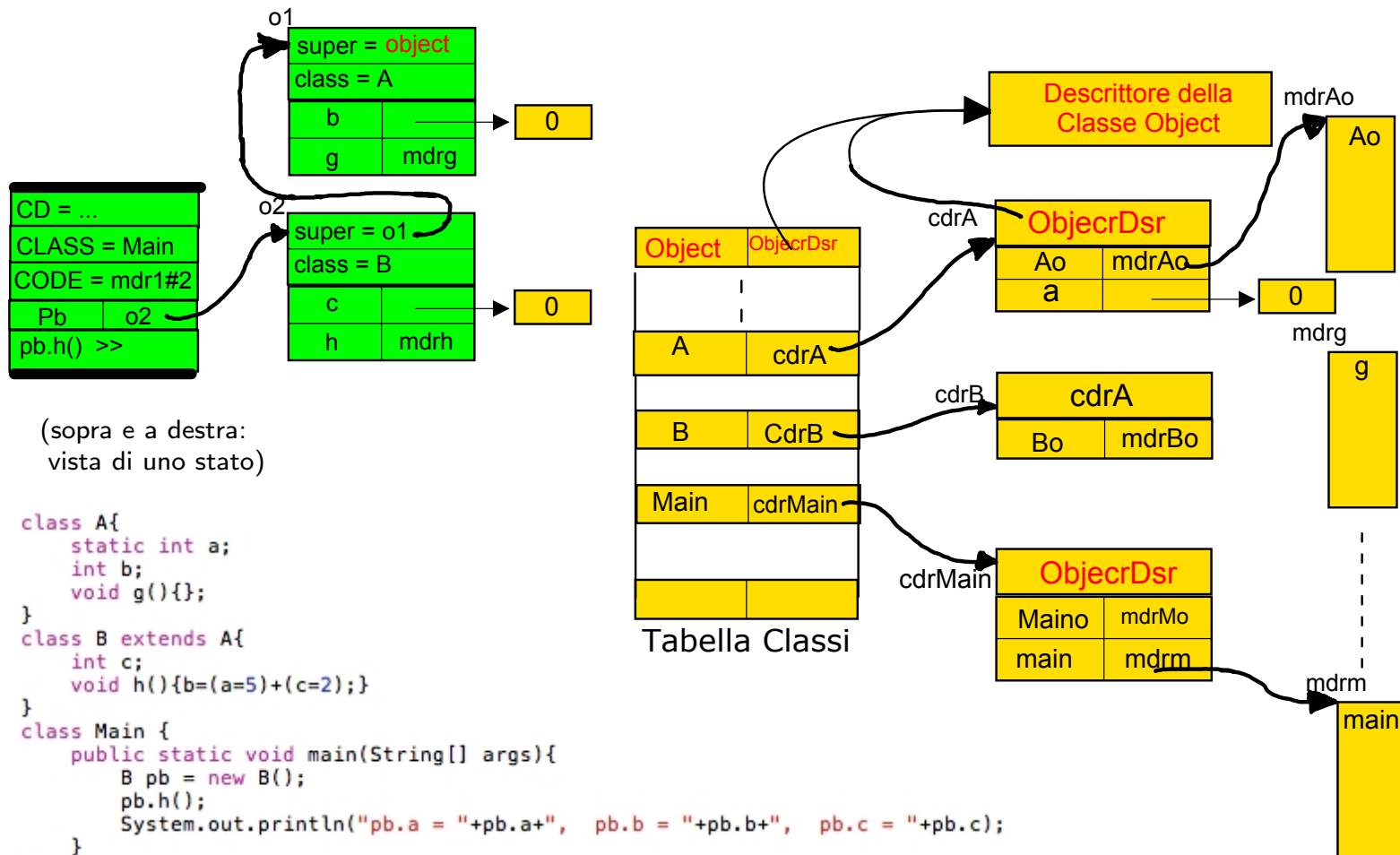
(a destra: creazione e vista di un oggetto di tipo B)

```
class A{
    static int a;
    int b;
    void g(){};
}
class B extends A{
    int c;
    void h(){b=(a=5)+(c=2);}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        pb.h();
        System.out.println("pb.a = "+pb.a+", pb.b = "+pb.b+", pb.c = "+pb.c);
    }
}
/*
Marco-Bellias-MacBook-Pro:sottoclasse marcob$ javac shadowing.java
Marco-Bellias-MacBook-Pro:sottoclasse marcob$ java Main
pb.a = 5, pb.b = 7, pb.c = 2
Marco-Bellias-MacBook-Pro:sottoclasse marcob$
*/
```



Classi e Sottoclassi: Oggetti e Costruttori

- **Sottoclasse** Estende la gerarchia delle classi del programma:



Marco-Bellias-MacBook-Pro:sottoclasse marcob\$ javac shadowing.java

Esercizi e : Osservazioni

Consideriamo la scrittura:

```
System.out.println("pb.a = "+ pb.a + ", pb.b = "+ pb..b + ", pb.c = "+ pb..c);
```

utilizzata nel metodo main per la stampa.

Esercizio

Cosa indica System:

*È una **keyword** di Java?*

*È una **struttura** procedurale e in tal caso: Comando, Dichiarazione, Espressione?*

*È una **classe** e in tal caso: Pubblica, Primitiva di Java, Definita dal Programma?*

*È un **object** e in tal caso: Primitivo di Java, Definito dal Programma?*

*È un **campo** e in tal caso di quale tipo e di qualche classe od oggetto?*

*È un **metodo** di qualche classe od Oggetto e in tal caso di quale?*

Esercizio

Cosa indicano out: e println, rispettivamente

*È una **keyword** di Java?*

*È una **struttura** procedurale e in tal caso: Comando, Dichiarazione, Espressione?*

*È una **classe** e in tal caso: Pubblica, Primitiva di Java, Definita dal Programma?*

*È un **object** e in tal caso: Primitivo di Java, Definito dal Programma?*

*È un **campo** e in tal caso di quale tipo e di qualche classe od oggetto?*

*È un **metodo** di qualche classe od Oggetto e in tal caso di quale?*

Esercizi e :Osservazioni

Consideriamo la scrittura sotto che inseriremo come ultimo statement di main, sopra:

```
System.out.println("pb = " + pb);
```

Esercizio

Si dica:

Cosa ci aspettiamo che calcoli ?

Si verifichi cosa effettivamente produce

Si commenti la differenza tra quanto atteso e quanto ottenuto. In particolare si motivi la scrittura prodotta.

Esercizio

Si dica cos'è il metodo toString e in particolare:

*È una **keyword** di Java?*

*È una **struttura** procedurale e in tal caso: Comando, Dichiarazione, Espressione?*

*È una **classe** e in tal caso: Pubblica, Primitiva di Java, Definita dal Programma?*

*È un **object** e in tal caso: Primitivo di Java, Definito dal Programma?*

*È un **campo** e in tal caso di quale tipo e di qualche classe od oggetto?*

*È un **metodo** di qualche classe od Oggetto e in tal caso di quale?*

Esercizio

- Si definisca una presentazione per gli oggetti di tipo A e di tipo B

- Si mostri come modificare il metodo main per stampare la presentazione dell'oggetto assegnato alla variabile pb.

4/19

Ereditarietà: Shadowing

- **shadowing** Una sottoclasse ridefinisce un field, e accede di default ...

```
import java.io.*;
import java.util.*;

class A{
    static int a;
    int b;
    void g(){};
}
class B extends A{
    static int a;
    int c;
    void h(){b=(a=5)+(c=2);}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        pb.h();
        System.out.println("A.a = "+A.a+" , B.a = "+B.a);
    }
}
/*
bellia:shadowing marcobellia$ cd code
bellia:code marcobellia$ java Main
A.a = 0 , B.a = 5
Marco-Bellias-MacBook-Pro:sottoclasse marcob$
*/
```


Ereditarietà: Shadowing/2

- **shadowing** Una sottoclasse ridefinisce un field, accede di default ... Ma può accedere a entrambi. In questo caso, lo può fare in due modi diversi:

```
import java.io.*;
import java.util.*;

class A{
    static int a;
    int b;
    void g(){};
}
class B extends A{
    static int a;
    int c;
    void h(){b=(a=5)+(c=2)+(A.a=15);}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        pb.h();
        System.out.println("A.a = "+A.a+" , B.a = "+B.a);
    }
}
```

- Applicabile perchè il field è di classe.
- Il secondo modo, accede al campo attraverso l'oggetto

Ereditarietà: Shadowing/3

- **shadowing** Una sottoclasse ridefinisce un field, accede di default ... Ma può accedere a entrambi, in due modi diversi:

```
import java.io.*;
import java.util.*;

class A{
    static int a;
    int b;
    void g(){};
}
class B extends A{
    static int a;
    int c;
    void h(){b=(a=5)+(c=2)+(super.a=15);}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        pb.h();
        System.out.println("A.a = "+A.a+" , B.a = "+B.a);
    }
}
```

- Applicabile perchè il field è di classe.
- Il secondo modo, accede al campo attraverso l'oggetto

Ereditarietà: Overriding

- E l'analogo dello shadowing su metodi anziché campi.
- Una sottoclasse ridefinisce un metodo della super, rispettando le seguenti 3 condizioni:
 - metodo di istanza (non si applica a costruttori)
 - stesso nome, stessi tipi degli argomenti,
 - eventuale tipo del valore calcolato ed eventuali tipi delle eccezioni sollevabili devono essere sotto-tipi del metodo della super.

```
class Point {
    double x;
    Point(double n1){
        x = n1;
    }
    public double distance (Point p){
        return Math.abs(x-p.x);
    }
}
class D2Point extends Point{
    double y;
    D2Point(double n1, double n2){
        super(n1);
        y = n2;
    }
    public double distance (Point p){
        double u = y-((D2Point)p).y;
        double d = super.distance(p);
        return Math.sqrt(d*d+u*u);
    }
}
```

- Riutilizzo di codice.

Ereditarietà: Overriding e Late Binding

- Quale binding deve assegnare il compilatore all'identificatore "f" che compare nell'invocazione "aa.f()"?

```
import java.io.*;
import java.util.*;

class A{
    static int a;
    char c;
    void g(){};
    void f(){System.out.println("sono oggetto di classe effettiva A");}
}

class B extends A{
    static int a;
    int b;
    void h(){B.a=5+(a=3)};
    void f(){System.out.println("sono oggetto di classe effettiva B");}
}

class Main {
    public static void main(String[] args){
        B pb = new B();
        A pa = new A();
        A aa = pb; //ad aa posso assegnare sia pb sia pa
        aa.f();
        ((B)aa).h();
        System.out.println("A.a = "+A.a+", B.a = "+B.a);
    }
}
```

- **Late Binding** Il binding è stabilito dinamicamente, guardando:
 - ◇ il tipo effettivo dell'oggetto calcolato dall'espressione di invocazione "aa" nel nostro caso.

Ereditarietà: Overriding e (Down) Cast

- La variabile "aa" ha comunque, tipo A e l'espressione "aa.h()" non è (sempre) definita.

```
import java.io.*;
import java.util.*;

class A{
    static int a;
    char c;
    void g(){};
    void f(){System.out.println("sono il metodo f di A");}
}
class B extends A{
    static int a;
    int b;
    void h(){B.a=5+(a=3)};
    void f(){System.out.println("sono il metodo f di B");}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        A pa = new A();
        A aa = pb;//ad aa posso assegnare sia pb sia pa
        aa.f();
        //aa.h(); -- error: cannot find symbol h
        ((B)aa).h(); //il cast rimanda il controllo al tempo di esecuzione
        System.out.println("A.a = "+A.a+", B.a = "+B.a);
    }
}
/*
Marco-Bellias-MacBook-Pro:overridingLateBinding marcob$ java Main
sono il metodo f di B
A.a = 0, B.a = 8
Marco-Bellias-MacBook-Pro:overridingLateBinding marcob$
*/
```

Ereditarietà: Overriding e (Down) Cast/2

- **(Down) Cast** deve essere utilizzato: (T)E "dichiara" che il tipo effettivo del valore calcolato da E sia T.

```
import java.io.*;
import java.util.*;

class A{
    static int a;
    char c;
    void g(){};
    void f(){System.out.println("sono il metodo f di A");}
}
class B extends A{
    static int a;
    int b;
    void h(){B.a=5+(a=3)};
    void f(){System.out.println("sono il metodo f di B");}
}
class Main {
    public static void main(String[] args){
        B pb = new B();
        A pa = new A();
        A aa = pa;//ad aa posso assegnare sia pb sia pa
        aa.f();
        //aa.h(); -- error: cannot find symbol h
        ((B)aa).h(); //il cast rimanda il controllo al tempo di esecuzione
        System.out.println("A.a = "+A.a+", B.a = "+B.a);
    }
}
/*
Marco-Bellias-MacBook-Pro:overridingLateBinding marcob$ java Main
sono il metodo f di A
Exception in thread "main" java.lang.ClassCastException: A cannot be cast to B
at Main.main(staticE.java:25)
Marco-Bellias-MacBook-Pro:overridingLateBinding marcob$
*/
```

12/17

Overloading vs. Overriding

- **Overloading.** Metodi statici e non, ereditati e non, che hanno stesso nome ma a coppie, signature differenti:
 - per numero di argomenti, o per tipo di un argomento, oppure
 - se uno è ereditato, l'altro ha tipo del valore calcolato che **non** è **sottotipo** dell'ereditato.
- Metodi overloaded sono tutti visibili ed applicabili;
- Nell'invocazione di metodo overloaded, a compile time, è scelto quello tra gli applicabili più prossimo al tipo atteso;
- Cautela. Errori nella definizione di un metodo overridden, rendono il metodo della superclasse un m. overloaded ed applicabile quando invece ci si attendeva che fosse "scavalcato".

Overloading: Metodo Invocato

- **Overloading.** Metodi statici e non, ereditati e non, che hanno stesso nome ma a coppie, signature differenti:
 - per numero di argomenti, o per tipo di un argomento, oppure
 - se uno è ereditato, l'altro ha tipo del valore calcolato che **non** è **sottotipo** dell'ereditato.
- Nell'invocazione di metodo overloaded, a compile time, è scelto quello tra gli applicabili più prossimo al tipo atteso;

```
class A{}
class B extends A{}
class C extends B{}

class E{
    void over(A x, A y){//overloaded
        System.out.println("sono overAA di E");
    }
    void over(A x, B y){//overloaded
        System.out.println("sono overAB di E");
    }
    void over(B x, C y){//overloaded
        System.out.println("sono overBC di E");
    }
}
class Main{
    public static void main(String[] argv){
        A x = new A();
        B y1 = new B();
        B y2 = new B();
        new E().over(y1,y2);
    }
}
```


Overloading: Rischi

- Cautela. Errori nella definizione di un metodo overridden, rendono il metodo della superclasse un metodo overloaded ed applicabile ...

```
package D2PointEx1;

import java.io.*;
import java.util.*;

class Point {
    double x;
    Point(double n1){
        x = n1;
    }
    public double distance (Point p){
        return Math.abs(x-p.x);
    }
}

class D2Point extends Point{
    double y;
    D2Point(double n1, double n2){
        super(n1);
        y = n2;
    }
    public double distance (D2Point p){
        double u = y-((D2Point)p).y;
        double d = super.distance(p);
        return Math.sqrt(d*d+u*u);
    }
}

class Main {
    public static void main(String[] args){
        Point p = new Point(3);
        D2Point q = new D2Point(0,3);
        System.out.println("come si comporta la valutazione di q.distance(p)? ");
        System.out.println("Calcola il valore: " + q.distance(p));
    }
}
```

Esercizi

- Si consideri la struttura di classi in slide "Overloading: Metodo Invocato". Si dica se e come cambierebbe il comportamento di "Main.main" se apportiamo le seguenti modifiche:
 - linea 6: "A" invece di "void"
 - linea 8: inseriamo linea: "return new A();"
 - linea 9: "B" invece di "void"
 - linea 11: inseriamo linea: "return new B();"
 - linea 22: inseriamo linea: "new E().over(y1,new C());"Si giustifichi il comportamento ottenuto. In particolare si dica quali metodi risultano ancora overloaded?
- Si consideri la struttura di classi in slide "Overloading: Rischi". Si sostituisca in linea 21, il parametro "Point p" con D2Point p.
 - (1) Si dica se e come cambiano le proprietà dei metodi distance dichiarati in Point e D2Point;
 - (2) Si dica se e come cambia il comportamento del metodo Main.main;
 - (3) Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di Main.main. Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale computazione.
- Si consideri la struttura di classi in "Overloading: Rischi". Si sostituisca nell'ultima linea di Main.main, l'espressione "q.distance(p)" con "p.distance(q)". Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di "p.distance(q)". Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale valutazione.
- Si consideri la struttura di classi in slide "Overloading: Rischi". Si dica se e come cambierebbe il comportamento di Main.main, allorchè la sua ultima linea avesse "q.distance(p)" sostituita con "p.distance(q)". Si motivino adeguatamente le ragioni dell'eventuale cambiamento.
- Si consideri la struttura di classi in slide "Overloading: Rischi". Si sostituisca nell'ultima linea di Main.main, l'espressione "q.distance(p)" con "p.distance(q)". Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di "p.distance(q)". Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale valutazione.
- Si consideri la struttura di classi in "Overloading: Rischi". Si sostituisca nella prima linea di Main.main, l'espressione "new Point(3)" con "new D2Point(3,-1)". Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di "q.distance(p)". Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale valutazione.

Altri Esercizi in EserciziL13.pdf

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↻ 17/17

Esercizi

- Si consideri la struttura di classi in slide "Overloading: Metodo Invocato". Si dica se e come cambierebbe il comportamento di "Main.main" se apportiamo le seguenti modifiche:
 - linea 6: "A" invece di "void"
 - linea 8: inseriamo linea: "return new A();"
 - linea 9: "B" invece di "void"
 - linea 11: inseriamo linea: "return new B();"
 - linea 22: inseriamo linea: "new E().over(y1,new C());"Si giustifichi il comportamento ottenuto. In particolare si dica quali metodi risultano ancora overloaded?
- Si consideri la struttura di classi in slide "Overloading: Rischi". Si sostituisca in linea 21, il parametro `D2Point p` con `Point p`.
 - (1) Si dica se e come cambiano le proprietà dei metodi `distance` dichiarati in `Point` e `D2Point`;
 - (2) Si dica se e come cambia il comportamento del metodo `Main.main`;
 - (3) Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di `Main.main`. Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale computazione.
- Si consideri la struttura di classi in "Overloading: Rischi". Si sostituisca nell'ultima linea di `Main.main`, l'espressione "`q.distance(p)`" con "`p.distance(q)`". Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di "`p.distance(q)`". Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale valutazione.
- Si consideri la struttura di classi in slide "Overloading: Rischi". Si dica se e come cambierebbe il comportamento di `Main.main`, allorchè la sua ultima linea avesse "`q.distance(p)`" sostituita con "`p.distance(q)`". Si motivino adeguatamente le ragioni dell'eventuale cambiamento.
- Si consideri la struttura di classi in "Overloading: Rischi". Si sostituisca nella prima linea di `Main.main`, l'espressione "`new Point(3)`" con "`new D2Point(3,-1)`". Si mostri l'evoluzione dello Stack di Controllo e dell'Heap durante la computazione di "`q.distance(p)`". Allo scopo si mostri anzitutto lo stato assunto immediatamente prima di tale valutazione.

Altri Esercizi in [EserciziL13.pdf](#)